Security

Trident

NetApp
June 30, 2026

Table of Contents

Security

Security

Use the recommendations listed here to ensure your Trident installation is secure.

Run Trident in its own namespace

It is important to prevent applications, application administrators, users, and management applications from accessing Trident object definitions or the pods to ensure reliable storage and block potential malicious activity.

To separate the other applications and users from Trident, always install Trident in its own Kubernetes namespace (`trident`). Putting Trident in its own namespace assures that only the Kubernetes administrative personnel have access to the Trident pod and the artifacts (such as backend and CHAP secrets if applicable) stored in the namespaced CRD objects.
You should ensure that you allow only administrators access to the Trident namespace and thus access to the `tridentctl` application.

Use CHAP authentication with ONTAP SAN backends

Trident supports CHAP-based authentication for ONTAP SAN workloads (using the `ontap-san` and `ontap-san-economy` drivers). NetApp recommends using bidirectional CHAP with Trident for authentication between a host and the storage backend.

For ONTAP backends that use the SAN storage drivers, Trident can set up bidirectional CHAP and manage CHAP usernames and secrets through `tridentctl`.
Refer to Prepare to configure backend with ONTAP SAN drivers to understand how Trident configures CHAP on ONTAP backends.

Use CHAP authentication with NetApp HCI and SolidFire backends

NetApp recommends deploying bidirectional CHAP to ensure authentication between a host and the NetApp HCI and SolidFire backends. Trident uses a secret object that includes two CHAP passwords per tenant. When Trident is installed, it manages the CHAP secrets and stores them in a `tridentvolume` CR object for the respective PV. When you create a PV, Trident uses the CHAP secrets to initiate an iSCSI session and communicate with the NetApp HCI and SolidFire system over CHAP.

The volumes that are created by Trident are not associated with any Volume Access Group.

Use Trident with NVE and NAE

NetApp ONTAP provides data-at-rest encryption to protect sensitive data in the event a disk is stolen, returned, or repurposed. For details, refer to Configure NetApp Volume Encryption overview.

- If NAE is enabled on the backend, any volume provisioned in Trident will be NAE-enabled.
  - You can set the NVE encryption flag to `""` to create NAE-enabled volumes.
- If NAE is not enabled on the backend, any volume provisioned in Trident will be NVE-enabled unless the NVE encryption flag is set to `false` (the default value) in the backend configuration.

Volumes created in Trident on an NAE-enabled backend must be NVE or NAE encrypted.

- You can set the NVE encryption flag to `true` in the Trident backend configuration to override the NAE encryption and use a specific encryption key on a per volume basis.
- Setting the NVE encryption flag to `false` on an NAE-enabled backend creates an NAE-enabled volume. You cannot disable NAE encryption by setting the NVE encryption flag to `false`.

- You can manually create an NVE volume in Trident by explicitly setting the NVE encryption flag to `true`.

For more information on backend configuration options, refer to:

- ONTAP SAN configuration options
- ONTAP NAS configuration options

Linux Unified Key Setup (LUKS)

You can enable Linux Unified Key Setup (LUKS) to encrypt ONTAP SAN and ONTAP SAN ECONOMY volumes on Trident. Trident supports passphrase rotation and volume expansion for LUKS-encrypted volumes.

In Trident, LUKS-encrypted volumes use the aes-xts-plain64 cypher and mode, as recommended by NIST.

LUKS encryption is not supported for ASA r2 systems. For information about ASA r2 systems, see Learn about ASA r2 storage systems.

Before you begin

- Worker nodes must have cryptsetup 2.1 or higher (but lower than 3.0) installed. For more information, visit Gitlab: cryptsetup.
- For performance reasons, NetApp recommends that worker nodes support Advanced Encryption Standard New Instructions (AES-NI). To verify AES-NI support, run the following command:

```
grep "aes" /proc/cpuinfo
```

If nothing is returned, your processor does not support AES-NI. For more information on AES-NI, visit: Intel: Advanced Encryption Standard Instructions (AES-NI).

Enable LUKS encryption

You can enable per-volume, host-side encryption using Linux Unified Key Setup (LUKS) for ONTAP SAN and ONTAP SAN ECONOMY volumes.

Steps

1. Define LUKS encryption attributes in the backend configuration. For more information on backend configuration options for ONTAP SAN, refer to ONTAP SAN configuration options.

```
{
  "storage": [
    {
      "labels": {
        "luks": "true"
      },
      "zone": "us_east_1a",
      "defaults": {
        "luksEncryption": "true"
      }
    },
    {
      "labels": {
        "luks": "false"
      },
      "zone": "us_east_1a",
      "defaults": {
        "luksEncryption": "false"
      }
    }
  ]
}
```

2. Use `parameters.selector` to define the storage pools using LUKS encryption. For example:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: luks
provisioner: csi.trident.netapp.io
parameters:
  selector: "luks=true"
  csi.storage.k8s.io/node-stage-secret-name: luks-${pvc.name}
  csi.storage.k8s.io/node-stage-secret-namespace: ${pvc.namespace}
```

3. Create a secret that contains the LUKS passphrase. For example:

```
kubectl -n trident create -f luks-pvc1.yaml
apiVersion: v1
kind: Secret
metadata:
  name: luks-pvc1
stringData:
  luks-passphrase-name: A
  luks-passphrase: secretA
```

Limitations

LUKS-encrypted volumes cannot take advantage of ONTAP deduplication and compression.

Backend configuration for importing LUKS volumes

To import a LUKS volume, you must set `luksEncryption` to `true` on the backend. The `luksEncryption` option tells Trident if the volume is LUKS-compliant (`true`) or not LUKS-compliant (`false`) as shown in the following example.

```
version: 1
storageDriverName: ontap-san
managementLIF: 10.0.0.1
dataLIF: 10.0.0.2
svm: trident_svm
username: admin
password: password
defaults:
  luksEncryption: 'true'
  spaceAllocation: 'false'
  snapshotPolicy: default
  snapshotReserve: '10'
```

PVC configuration for importing LUKS volumes

To import LUKS volumes dynamically, set the annotation `trident.netapp.io/luksEncryption` to `true` and include a LUKS-enabled storage class in the PVC as shown in this example.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: luks-pvc
  namespace: trident
  annotations:
    trident.netapp.io/luksEncryption: "true"
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: luks-sc
```

Rotate a LUKS passphrase

You can rotate the LUKS passphrase and confirm rotation.

> Do not forget a passphrase until you have verified it is no longer referenced by any volume, snapshot, or secret. If a referenced passphrase is lost, you might be unable to mount the volume and the data will remain encrypted and inaccessible.

About this task

LUKS passphrase rotation occurs when a pod that mounts the volume is created after a new LUKS passphrase is specified. When a new pod is created, Trident compares the LUKS passphrase on the volume to the active passphrase in the secret.

- If the passphrase on the volume does not match the active passphrase in the secret, rotation occurs.
- If the passphrase on the volume matches the active passphrase in the secret, the `previous-luks-passphrase` parameter is ignored.

Steps

1. Add the `node-publish-secret-name` and `node-publish-secret-namespace` StorageClass parameters. For example:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: csi-san
provisioner: csi.trident.netapp.io
parameters:
  trident.netapp.io/backendType: "ontap-san"
  csi.storage.k8s.io/node-stage-secret-name: luks
  csi.storage.k8s.io/node-stage-secret-namespace: ${pvc.namespace}
  csi.storage.k8s.io/node-publish-secret-name: luks
  csi.storage.k8s.io/node-publish-secret-namespace: ${pvc.namespace}
```

2. Identify existing passphrases on the volume or snapshot.

Volume

```
tridentctl -d get volume luks-pvc1
GET http://127.0.0.1:8000/trident/v1/volume/<volumeID>

...luksPassphraseNames:["A"]
```

Snapshot

```
tridentctl -d get snapshot luks-pvc1
GET http://127.0.0.1:8000/trident/v1/volume/<volumeID>/<snapshotID>

...luksPassphraseNames:["A"]
```

3. Update the LUKS secret for the volume to specify the new and previous passphrases. Ensure `previous-luke-passphrase-name` and `previous-luks-passphrase` match the previous passphrase.

```
apiVersion: v1
kind: Secret
metadata:
  name: luks-pvc1
stringData:
  luks-passphrase-name: B
  luks-passphrase: secretB
  previous-luks-passphrase-name: A
  previous-luks-passphrase: secretA
```

4. Create a new pod mounting the volume. This is required to initiate the rotation.
5. Verify the the passphrase was rotated.

Volume

```
tridentctl -d get volume luks-pvc1
GET http://127.0.0.1:8000/trident/v1/volume/<volumeID>

...luksPassphraseNames:["B"]
```

Snapshot

```
tridentctl -d get snapshot luks-pvc1
GET http://127.0.0.1:8000/trident/v1/volume/<volumeID>/<snapshotID>

...luksPassphraseNames:["B"]
```

Results

The passphrase was rotated when only the new passphrase is returned on the volume and snapshot.

If two passphrases are returned, for example `luksPassphraseNames: ["B", "A"]`, the rotation is incomplete. You can trigger a new pod to attempt to complete the rotation.

Enable volume expansion

You can enable volume expansion on a LUKS-encrypted volume.

Steps

1. Enable the `CSINodeExpandSecret` feature gate (beta 1.25+). Refer to Kubernetes 1.25: Use Secrets for Node-Driven Expansion of CSI Volumes for details.
2. Add the `node-expand-secret-name` and `node-expand-secret-namespace` StorageClass parameters. For example:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: luks
provisioner: csi.trident.netapp.io
parameters:
  selector: "luks=true"
  csi.storage.k8s.io/node-stage-secret-name: luks-${pvc.name}
  csi.storage.k8s.io/node-stage-secret-namespace: ${pvc.namespace}
  csi.storage.k8s.io/node-expand-secret-name: luks-${pvc.name}
  csi.storage.k8s.io/node-expand-secret-namespace: ${pvc.namespace}
allowVolumeExpansion: true
```

Results

When you initiate online storage expansion, the kubelet passes the appropriate credentials to the driver.

Kerberos in-flight encryption

Using Kerberos in-flight encryption, you can improve data access security by enabling encryption for the traffic between your managed cluster and the storage backend.

Trident supports Kerberos encryption for ONTAP as a storage backend:

- **On-premise ONTAP** - Trident supports Kerberos encryption over NFSv3 and NFSv4 connections from Red Hat OpenShift and upstream Kubernetes clusters to on-premise ONTAP volumes.

You can create, delete, resize, snapshot, clone, read-only clone, and import volumes that use NFS encryption.

Configure in-flight Kerberos encryption with on-premise ONTAP volumes

You can enable Kerberos encryption on the storage traffic between your managed cluster and an on-premise ONTAP storage backend.

> Kerberos encryption for NFS traffic with on-premise ONTAP storage backends is only supported using the `ontap-nas` storage driver.

Before you begin

- Ensure that you have access to the `tridentctl` utility.
- Ensure you have administrator access to the ONTAP storage backend.
- Ensure you know the name of the volume or volumes you will be sharing from the ONTAP storage backend.
- Ensure that you have prepared the ONTAP storage VM to support Kerberos encryption for NFS volumes. Refer to Enable Kerberos on a dataLIF for instructions.
- Ensure that any NFSv4 volumes you use with Kerberos encryption are configured correctly. Refer to the NetApp NFSv4 Domain Configuration section (page 13) of the NetApp NFSv4 Enhancements and Best Practices Guide.

Add or modify ONTAP export policies

You need to add rules to existing ONTAP export policies or create new export polices that support Kerberos encryption for the ONTAP storage VM root volume as well as any ONTAP volumes shared with the upstream Kubernetes cluster. The export policy rules you add, or new export policies you create, need to support the following access protocols and access permissions:

Access protocols

Configure the export policy with NFS, NFSv3, and NFSv4 access protocols.

Access details

You can configure one of three different versions of Kerberos encryption, depending on your needs for the volume:

- **Kerberos 5** - (authentication and encryption)
- **Kerberos 5i** - (authentication and encryption with identity protection)

- **Kerberos 5p** - (authentication and encryption with identity and privacy protection)

Configure the ONTAP export policy rule with the appropriate access permissions. For example, if clusters will be mounting the NFS volumes with a mixture of Kerberos 5i and Kerberos 5p encryption, use the following access settings:

Type	Read-only access	Read/Write access	Superuser access
UNIX	Enabled	Enabled	Enabled
Kerberos 5i	Enabled	Enabled	Enabled
Kerberos 5p	Enabled	Enabled	Enabled

Refer to the following documentation for how to create ONTAP export policies and export policy rules:

- Create an export policy
- Add a rule to an export policy

Create a storage backend

You can create a Trident storage backend configuration that includes Kerberos encryption capability.

About this task

When you create a storage backend configuration file that configures Kerberos encryption, you can specify one of three different versions of Kerberos encryption using the `spec.nfsMountOptions` parameter:

- `spec.nfsMountOptions: sec=krb5` (authentication and encryption)
- `spec.nfsMountOptions: sec=krb5i` (authentication and encryption with identity protection)
- `spec.nfsMountOptions: sec=krb5p` (authentication and encryption with identity and privacy protection)

Specify only one Kerberos level. If you specify more than one Kerberos encryption level in the parameter list, only the first option is used.

Steps

1. On the managed cluster, create a storage backend configuration file using the following example. Replace values in brackets <> with information from your environment:

```
apiVersion: v1
kind: Secret
metadata:
  name: backend-ontap-nas-secret
type: Opaque
stringData:
  clientID: <CLIENT_ID>
  clientSecret: <CLIENT_SECRET>
---
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-ontap-nas
spec:
  version: 1
  storageDriverName: "ontap-nas"
  managementLIF: <STORAGE_VM_MGMT_LIF_IP_ADDRESS>
  dataLIF: <PROTOCOL_LIF_FQDN_OR_IP_ADDRESS>
  svm: <STORAGE_VM_NAME>
  username: <STORAGE_VM_USERNAME_CREDENTIAL>
  password: <STORAGE_VM_PASSWORD_CREDENTIAL>
  nasType: nfs
  nfsMountOptions: ["sec=krb5i"] #can be krb5, krb5i, or krb5p
  qtreesPerFlexvol:
  credentials:
    name: backend-ontap-nas-secret
```

2. Use the configuration file you created in the previous step to create the backend:

```
tridentctl create backend -f <backend-configuration-file>
```

If the backend creation fails, something is wrong with the backend configuration. You can view the logs to determine the cause by running the following command:

```
tridentctl logs
```

After you identify and correct the problem with the configuration file, you can run the create command again.

Create a storage class

You can create a storage class to provision volumes with Kerberos encryption.

About this task

When you create a storage class object, you can specify one of three different versions of Kerberos encryption using the `mountOptions` parameter:

- `mountOptions: sec=krb5` (authentication and encryption)
- `mountOptions: sec=krb5i` (authentication and encryption with identity protection)
- `mountOptions: sec=krb5p` (authentication and encryption with identity and privacy protection)

Specify only one Kerberos level. If you specify more than one Kerberos encryption level in the parameter list, only the first option is used. If the level of encryption you specified in the storage backend configuration is different than the level you specify in the storage class object, the storage class object takes precedence.

Steps

1. Create a StorageClass Kubernetes object, using the following example:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-nas-sc
provisioner: csi.trident.netapp.io
mountOptions:
  - sec=krb5i #can be krb5, krb5i, or krb5p
parameters:
  backendType: ontap-nas
  storagePools: ontapnas_pool
  trident.netapp.io/nasType: nfs
allowVolumeExpansion: true
```

2. Create the storage class:

```
kubectl create -f sample-input/storage-class-ontap-nas-sc.yaml
```

3. Make sure that the storage class has been created:

```
kubectl get sc ontap-nas-sc
```

   You should see output similar to the following:

```
NAME          PROVISIONER             AGE
ontap-nas-sc    csi.trident.netapp.io   15h
```

Provision volumes

After you create a storage backend and a storage class, you can now provision a volume. For instructions,

refer to Provision a volume.

Configure in-flight Kerberos encryption with Azure NetApp Files volumes

You can enable Kerberos encryption on the storage traffic between your managed cluster and a single Azure NetApp Files storage backend or a virtual pool of Azure NetApp Files storage backends.

Before you begin

- Ensure that you have enabled Trident on the managed Red Hat OpenShift cluster.
- Ensure that you have access to the `tridentctl` utility.
- Ensure that you have prepared the Azure NetApp Files storage backend for Kerberos encryption by noting the requirements and following the instructions in Azure NetApp Files documentation.
- Ensure that any NFSv4 volumes you use with Kerberos encryption are configured correctly. Refer to the NetApp NFSv4 Domain Configuration section (page 13) of the NetApp NFSv4 Enhancements and Best Practices Guide.

Create a storage backend

You can create an Azure NetApp Files storage backend configuration that includes Kerberos encryption capability.

About this task

When you create a storage backend configuration file that configures Kerberos encryption, you can define it so that it should be applied at one of two possible levels:

- The **storage backend level** using the `spec.kerberos` field
- The **virtual pool level** using the `spec.storage.kerberos` field

When you define the configuration at the virtual pool level, the pool is selected using the label in the storage class.

At either level, you can specify one of three different versions of Kerberos encryption:

- `kerberos: sec=krb5` (authentication and encryption)
- `kerberos: sec=krb5i` (authentication and encryption with identity protection)
- `kerberos: sec=krb5p` (authentication and encryption with identity and privacy protection)

Steps

1. On the managed cluster, create a storage backend configuration file using one of the following examples, depending on where you need to define the storage backend (storage backend level or virtual pool level). Replace values in brackets <> with information from your environment:

Storage backend level example

```
apiVersion: v1
kind: Secret
metadata:
  name: backend-tbc-secret
type: Opaque
stringData:
  clientID: <CLIENT_ID>
  clientSecret: <CLIENT_SECRET>

---
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc
spec:
  version: 1
  storageDriverName: azure-netapp-files
  subscriptionID: <SUBSCRIPTION_ID>
  tenantID: <TENANT_ID>
  location: <AZURE_REGION_LOCATION>
  serviceLevel: Standard
  networkFeatures: Standard
  capacityPools: <CAPACITY_POOL>
  resourceGroups: <RESOURCE_GROUP>
  netappAccounts: <NETAPP_ACCOUNT>
  virtualNetwork: <VIRTUAL_NETWORK>
  subnet: <SUBNET>
  nasType: nfs
  kerberos: sec=krb5i #can be krb5, krb5i, or krb5p
  credentials:
    name: backend-tbc-secret
```

Virtual pool level example

```
---
apiVersion: v1
kind: Secret
metadata:
  name: backend-tbc-secret
type: Opaque
stringData:
  clientID: <CLIENT_ID>
  clientSecret: <CLIENT_SECRET>

---
apiVersion: trident.netapp.io/v1
kind: TridentBackendConfig
metadata:
  name: backend-tbc
spec:
  version: 1
  storageDriverName: azure-netapp-files
  subscriptionID: <SUBSCRIPTION_ID>
  tenantID: <TENANT_ID>
  location: <AZURE_REGION_LOCATION>
  serviceLevel: Standard
  networkFeatures: Standard
  capacityPools: <CAPACITY_POOL>
  resourceGroups: <RESOURCE_GROUP>
  netappAccounts: <NETAPP_ACCOUNT>
  virtualNetwork: <VIRTUAL_NETWORK>
  subnet: <SUBNET>
  nasType: nfs
  storage:
    - labels:
        type: encryption
      kerberos: sec=krb5i #can be krb5, krb5i, or krb5p
  credentials:
    name: backend-tbc-secret
```

2. Use the configuration file you created in the previous step to create the backend:

```
tridentctl create backend -f <backend-configuration-file>
```

If the backend creation fails, something is wrong with the backend configuration. You can view the logs to determine the cause by running the following command:

```
tridentctl logs
```

After you identify and correct the problem with the configuration file, you can run the create command again.

Create a storage class

You can create a storage class to provision volumes with Kerberos encryption.

Steps

1. Create a StorageClass Kubernetes object, using the following example:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: sc-nfs
provisioner: csi.trident.netapp.io
parameters:
  backendType: azure-netapp-files
  trident.netapp.io/nasType: nfs
  selector: type=encryption
```

2. Create the storage class:

```
kubectl create -f sample-input/storage-class-sc-nfs.yaml
```

3. Make sure that the storage class has been created:

```
kubectl get sc -sc-nfs
```

You should see output similar to the following:

```
NAME          PROVISIONER             AGE
sc-nfs        csi.trident.netapp.io   15h
```

Provision volumes

After you create a storage backend and a storage class, you can now provision a volume. For instructions, refer to Provision a volume.

Copyright information

Trademark information